

CreatingCustomTranslators

NOTE: This is outdated information that applies only to Tapestry 4. For current information see <http://tapestry.apache.org/input-validation.html>.

Creating a Translator is very similar to [CreatingCustomValidators](#).

1. Extend Abstract Translator

The important part is to implement the formatObject and parseText methods to convert between your object and string.

In my example I have a boolean object that is the gender of someone represented by the 1 character strings M and F, defaulting to M when undefined.

```

package uk.org.chandlerfamily.tapestry.translators;

import java.util.Locale;

import org.apache.tapestry.form.IFormComponent;
import org.apache.tapestry.form.ValidationMessages;
import org.apache.tapestry.form.translator.AbstractTranslator;
import org.apache.tapestry.valid.ValidatorException;

public class Gender extends AbstractTranslator
{

    private String _parameter;

    public Gender()
    {
        super();
    }

    public Gender(String initializer)
    {
        super(initializer);
    }

    public void setParameter(String parameter)
    {
        this._parameter = parameter;
    }

    public String getParameter()
    {
        return _parameter;
    }

    public String formatObject(IFormComponent arg0, Locale arg1, Object arg2)
    {
        if (arg2 == null)
            return "M";

        return (Boolean) arg2 ? "M" : "F";
    }

    public Object parseText(IFormComponent field, ValidationMessages messages,
                           String text) throws ValidatorException
    {

        String [] args = new String[1];
        args[0] = text;

        if (!text.equalsIgnoreCase("M") && !text.equalsIgnoreCase("F"))
        {
            throw new ValidatorException(messages.formatValidationMessage(getMessage(), "invalid-format", args));
        }

        return text.equalsIgnoreCase("M");
    }
}

```

Note: I haven't managed to figure out how to create new message keys. The invalid-format message key is a standard tapestry one.

2. Configure the translator in hivemodule.xml

Use the contribution element, with a bean element inside. You can put more than one bean element inside if you have multiple translators

```
<contribution configuration-id="tapestry.form.translator.Translators">
    <bean name="gender" class="uk.org.chandlerfamily.tapestry.translators.Gender" />
</contribution>
```

3. Use in your templates

use the translator:beanName to access the translator - with the value parameter being the translated object type.

```
<td><span jwcid="@FieldLabel"
    field="component:gender" >Gender:</span></td>
<td><input jwcid="gender@TextField"
    value="ognl:gender"
    size="1"
    translator="translator:gender"
    displayName="Gender" /></td>
```

and in your class - see how I have used a real method in an abstract class to create a default value of true for this parameter (not sure if this is the correct way to do it)

```
public abstract class Create extends BasePage {
    private boolean _gender = true; //gives a default value
    public boolean getGender() {
        return _gender;
    }
    public void setGender(boolean gender) {
        this._gender = gender;
    }
    ...
}
```

4. Configurable Translators

The translator factory relies on Hivemind's BeanFactory which enables passing a single String to a bean constructor. Implement a constructor which takes a single String as a parameter, and use the syntax

```
<binding name="translator" value="translator:gender,parameter"/>
```

This didn't work for me: I instead used the setter/getter ("setParameter()"/"getParameter()") in the translator class and the syntax

```
<binding name="translator" value="translator:gender,parameter=value"/>
```