How to order your feedback messages

The default implementation of FeedbackPanel displays messages unsorted.
You can sort messages any way you like by either setting the sortingComparator property of FeedbackPanel, or to provide a custom implementation of the FeedbackMessagesModel by overriding FeedbackPanel.getFeedbackMessagesModel().

As an example, let's say we want to sort the messages in the order that we added our components to a form. For this example, we alter the FormInput example of wicket-examples.

First, use a variable (like a list) to store your order:

```
private List componentOrder = new ArrayList();
```

Then, override method add to automatically add the order:

```
/**
* @see wicket.MarkupContainer#add(wicket.Component)
*/
public MarkupContainer add(Component component)
{
  super.add(component);
  componentOrder.add(component);
  return (MarkupContainer)component;
}
```

Now, create a comparator that uses this and set it on the FeedbackPanel:

```
feedback.setSortingComparator(new Comparator()
{
  public int compare(Object o1, Object o2)
  {
    FeedbackMessage m1 = (FeedbackMessage)o1;
    FeedbackMessage m2 = (FeedbackMessage)o2;
    int ix1 = componentOrder.indexOf(m1.getReporter());
    int ix2 = componentOrder.indexOf(m2.getReporter());
    return ix1 - ix2;
  }
});
```

That's it! The complete patched version of FormInput (based on v1.41):

```
package wicket.examples.forminput;

import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Locale;

import wicket.Component;
import wicket.FeedbackMessage;
import wicket.MarkupContainer;
import wicket.examples.WicketExamplePage;
import wicket.markup.html.form.CheckBox;
import wicket.markup.html.form.DropDownChoice;
import wicket.markup.html.form.Form;
import wicket.markup.html.form.ImageButton;
import wicket.markup.html.form.ListMultipleChoice;
import wicket.markup.html.form.RadioChoice;
import wicket.markup.html.form.RequiredTextField;
import wicket.markup.html.form.TextField;
import wicket.markup.html.form.model.ChoiceList;
import wicket.markup.html.form.model.IChoice;
```

```
import wicket.markup.html.form.validation.IntegerValidator;
import wicket.markup.html.image.Image;
import wicket.markup.html.link.Link;
import wicket.markup.html.panel.FeedbackPanel;
import wicket.model.CompoundPropertyModel;
import wicket.model.PropertyModel;
import wicket.protocol.http.WebRequest;
import wicket.util.convert.IConverter;

/**
 * Example for form input.
 *
 * @author Eelco Hillenius
 * @author Jonathan Locke
 */
public class FormInput extends WicketExamplePage
{
        /** Relevant locales wrapped in a list. */
        private static final List LOCALES = Arrays.asList(new Locale[]
                { Locale.US, new Locale("nl"), Locale.GERMANY , Locale.SIMPLIFIED_CHINESE });

        /** available numbers for the radio selection. */
        private static final List NUMBERS = Arrays.asList(new String[]{"1", "2", "3"});

        /** available sites for the multiple select. */
        private static final List SITES = Arrays.asList(
                        new String[]{"The Server Side", "Java Lobby", "Java.Net"});

        /**
         * Constructor
         */
        public FormInput()
        {
                Locale locale = getLocale();

                // Construct form and feedback panel and hook them up
                final FeedbackPanel feedback = new FeedbackPanel("feedback");
                add(feedback);
                add(new InputForm("inputForm", feedback));

                // Dropdown for selecting locale
                add(new LocaleDropDownChoice("localeSelect"));

                // Link to return to default locale
                add(new Link("defaultLocaleLink")
                {
                        public void onClick()
                        {
                                WebRequest request = (WebRequest)getRequest();
                                setLocale(request.getLocale());
                        }
                });
        }

        /**
         * Sets locale for the user's session (getLocale() is inherited from
         * Component)
         *
         * @param locale
         *            The new locale
         */
        public void setLocale(Locale locale)
        {
                getSession().setLocale(locale);
        }


        /**
         * Form for collecting input.
         */
        private class InputForm extends Form
```

```
{
	/**
	 * Keeps the add order for sorting messages.
	 */
	private List componentOrder = new ArrayList();

	/**
	 * Construct.
	 *
	 * @param name
	 *            Component name
	 * @param feedback
	 *            Feedback display for form
	 */
	public InputForm(String name, FeedbackPanel feedback)
	{
		super(name, new CompoundPropertyModel(new FormInputModel()), feedback);

		feedback.setSortingComparator(new Comparator()
				{
			public int compare(Object o1, Object o2)
			{
				FeedbackMessage m1 = (FeedbackMessage)o1;
				FeedbackMessage m2 = (FeedbackMessage)o2;
				int ix1 = componentOrder.indexOf(m1.getReporter());
				int ix2 = componentOrder.indexOf(m2.getReporter());
				return ix1 - ix2;
			}
		});

		add(new RequiredTextField("stringProperty"));
		add(new RequiredTextField("integerProperty", Integer.class));
		add(new RequiredTextField("doubleProperty", Double.class));
		add(new RequiredTextField("dateProperty", Date.class));
		add(new RequiredTextField("integerInRangeProperty", Integer.class).add(IntegerValidator
				.range(0, 100)));
		add(new CheckBox("booleanProperty"));
		add(new RadioChoice("numberRadioChoice", NUMBERS)
		{
			protected String getSuffix()
			{
				return "";
			}
		});
		add(new ListMultipleChoice("siteSelection", SITES));

		// as an example, we use a custom converter here.
		add(new TextField("urlProperty", URL.class)
		{
			public IConverter getConverter()
			{
				return new URLConverter();
			}
		});

		add(new ImageButton("saveButton"));

		add(new Link("resetButtonLink")
		{
			public void onClick()
			{
				// just call modelChanged so that any invalid input is cleared.
				InputForm.this.modelChanged();
			}
		}.add(new Image("resetButtonImage")));
	}

	/**
	 * @see wicket.markup.html.form.Form#onSubmit()
	 */
	public void onSubmit()
```

```
        {
                // Form validation successful. Display message showing edited model.
                info("Saved model " + getModelObject());
        }

        /**
         * @see wicket.MarkupContainer#add(wicket.Component)
         */
        public MarkupContainer add(Component component)
        {
                super.add(component);
                componentOrder.add(component);
                return (MarkupContainer)component;
        }
}

/**
 * Dropdown with Locales.
 */
private final class LocaleDropDownChoice extends DropDownChoice
{
        /**
         * Construct.
         * @param id component id
         */
        public LocaleDropDownChoice(String id)
        {
                super(id);

                // set the model that gets the current locale, and that is used for updating
                // the current locale to property 'locale' of FormInput
                setModel(new PropertyModel(FormInput.this, "locale"));

                // use a custom implementation of choices, as we want to display
                // the choices localized
                ChoiceList locales = new ChoiceList(LOCALES)
                {
                        protected IChoice newChoice(Object object, int index)
                        {
                                return new LocaleChoice((Locale)object, index);
                        }
                };
                setChoices(locales);
        }

        /**
         * @see wicket.markup.html.form.DropDownChoice#wantOnSelectionChangedNotifications()
         */
        protected boolean wantOnSelectionChangedNotifications()
        {
                // we want roundtrips when a the user selects another item
                return true;
        }

        /**
         * @see wicket.markup.html.form.DropDownChoice#onSelectionChanged(java.lang.Object)
         */
        public void onSelectionChanged(Object newSelection)
        {
                // note that we don't have to do anything here, as our property model allready
                // calls FormInput.setLocale when the model is updated
                // setLocale((Locale)newSelection); // so we don't need to do this
        }
}

/**
 * Choice for a locale.
 */
private final class LocaleChoice implements IChoice
{
        /** The index of the choice. */
```

```
                private final int index;

                /** The choice model object. */
                private final Locale locale;

                /**
                 * Constructor.
                 * @param locale The locale
                 * @param index The index of the object in the choice list
                 */
                public LocaleChoice(final Locale locale, final int index)
                {
                        this.locale = locale;
                        this.index = index;
                }

                /**
                 * @see wicket.markup.html.form.model.IChoice#getDisplayValue()
                 */
                public String getDisplayValue()
                {
                        String display = locale.getDisplayName(getLocale());
                        return display;
                }

                /**
                 * @see wicket.markup.html.form.model.IChoice#getId()
                 */
                public String getId()
                {
                        return Integer.toString(index);
                }

                /**
                 * @see wicket.markup.html.form.model.IChoice#getObject()
                 */
                public Object getObject()
                {
                        return locale;
                }
        }
}
```