SweenJet 1.0 - Developer Manual

Julius E. Adorf

Table of Contents

About SweenJet

Introduction

SweenJet is a open-source project hosted by SourceForge. It is a library that helps developers to create Java applications. It provides constrained models, useful Swing components (e.g. splash screen, input fields), actions, look-and-feel enhancements (e.g. themes, multi-line tooltip), and a complete preferences system.

Target audience

SweenJet targets Java application developers.

About this manual

This manual gives an overview about the library and shows its basic function, but it is not intended to replace the Javadoc.

Installation instructions

Requirements

SweenJet requires Java 5.0 to run. If you do not have installed Java JDK 5.0 yet, please download it from http://java.sun.com/j2se/1.5.0/download.jsp. If you are an end user, the Java Runtime Environment (JRE) is sufficient.

Steps

Simply extract the contents of the compressed ZIP file to the desired location, and you are ready to go.

Actions

Applications almost always have in common a basic set of actions (help, about, preferences, exit, etc.)

About

Shows a window that displays HTML-based info text.

```
// Create some CSS-rules
String cssRules = "* { font-family: Arial; font-size: 11pt; }";
// Create the about action, specifying the application name.
Action aboutAction = new AboutAction("SweenJet",
                            getClass().getResourceAsStream("/sweenjet/about.html"),
                            cssRules);
```

Help

Shows a help window.

```
URL helpSetURL = getClass().getResource("/sweenjet/help/developer/developer.hs");
Action helpAction = new HelpAction(helpSetURL);
```

Contextual help

Shows a help window and displays a help topic that belongs to a certain component. If the selected component has no help ID, the default page is displayed.

```
URL helpSetURL = getClass().getResource("/sweenjet/help/developer/developer.hs");
Action helpAction = new CSHAction(helpSetURL);
```

Preferences

Displays a preferences window. The constructor takes a component that enables the user to modify the settings.

```
// MyAppPreferencesPane must be a subclass of javax.swing.JComponent
Action preferencesAction = new PreferencesAction(new MyAppPreferencesPane());
```

Exit

Calls SystemManager.systemExit().

```
Runnable doBeforeQuitting = new Runnable() {
    public void run() {
        // Do some stuff here (e.g. save the configuration)
        // ...
    }
};
Action exitAction = new Exit(doBeforeQuitting);
```

Properties

A property wraps an object that represents a certain value. A property may extend another property, a so-called "parent". A class that extends org.municware.prop.Property works perfectly as the model of a MVC-based swing component. SweenJet provides several components whose models are instances of class Property.

Attributes

Name

Serves as an identifier for the property. It should be unique with in the scope of an application. The name cannot be changed.

Value

The current value of the property. Unless specified explicitly (i.e. it is null), the value is set to the default value. The value must fulfill all conditions imposed by the constraints.

If you use inheritance (i.e. you specify a parental property), the value is retrieved with the following deterministic, recursive algorithm.

1. value (skip, unless specified)
2. default value (skip, unless specified)
3. parent value (skip, unless specified)

The parent value is retrieved in the same way.

A null value denotes that the value is unspecified.

Default value

The default value of the property. Unless specified explicitly (i.e. it is null), the default value is set to the value of the parent. The default value must, like the value, fulfill all conditions imposed by the constraints.

A null value denotes that the value is unspecified.

Parent

A property may have got a parental property. The parental property only influences the value, the default value and the constraints.

Constraints

Specifies the conditions a value must fulfill to be valid. The constraints are effectively immutable - they can be changed by subclasses but should stay unmodified outside the constructor.

The constraints can lead to problems when using inheritance. Let's say a property defines some constraints but its parent does not. Then, the parental property could deliver values that are invalid and cause an exception. To avoid this, the property must not define any constraint that restricts any value delivered by the parent. As this cannot be guaranteed, a property with a parent inherits the parental constraints.

Display name

Serves as a human-readable name.

Description

Describes the property.

Serialization

The property class should not be serialized, rather key-value pairs. These key-value pairs can be read again. But if the reading fails, the application should usually not crash because each property should have a valid default value.

Extending class Property

You only need to extend this class if you want to provide some extra (e.g. convenience) methods. For reasons concerning consistency, the subclasses are highly recommended to provide three constructors taking the following parameters:

- String name, String displayName, String description, E defaultValue
- String name, String displayName, String description, E defaultValue, Constraint<E>... constraints
- String name, String displayName, String description, E defaultValue, Property<E> parent

An example for a subclass is class BooleanProperty, which adds the method isTrue().

Constraints

A constraint constrains objects to fulfill specific conditions. Thus, a constraint is a sort of filters.

A constraint should usually be deterministic, i.e. no random-selection. When extending this class, it is recommended to correctly implement the equals method, as two constraints that have equal sets of accepted objects should be equal. Hence, equality is measured upon the results of the accept method.

A constraint must provide a description, which should describe the constraint as precise and concise as possible. The description targets users. For example. the description could be used by a user interfaces in order to show the user what sort of value is valid.

Constraints are used by SweenJet's property framework but can also be used in other application areas.

Preferences

The preferences system makes heavily use of properties and constraints. It provides a flexible solution for the recurrent preferences problem.

In fact, the preferences framework only adds some swing components, the residual functionality is already provided by the property API.

The provided property editors are not restricted to be used in a preferences system but can be applied anywhere in a GUI.

PropertyField

Displays a certain property or/and lets the user modify it.

It is important to know that editing values in a property field do not change the underlying property immediately (except autoApply is set to true). Instead, the values must be applied.

SweenJet comes with the following property fields:

- BooleanPropertyField (boolean)
- ColorPropertyField (java.awt.Color)
- FilePropertyField (java.io.File)
- StringPropertyField (String)
- URLPropertyField (java.net.URL)

Look-and-Feel enhancements

Themes

SweenJet provides basic support for UI themes. Such a theme can be defined with a simple properties file.

```
# General settings
foreground = rgb(53,0,204)
background = orange
font = Arial-PLAIN-11

# Fine tuning
TextArea.background = black
MenuItem.acceleratorFont = Arial-PLAIN-9
```

In order to install a UI theme, you must call the following methods at the very beginning of your application, preferably before any Swing components are initalized.

```
try {
    SwingUtils.applyUITheme(getClass().getResource("/sweenjet/orange_UI.properties"));
} catch (IOException ioe) {
    ioe.printStackTrace();
    SwingUtils.showErrorDialog(null, ioe, "Cannot load UI theme");
}
```

Tooltips

Swing tooltips are nice and often helpful. But often the description does not fit in one single line so that you are tempted to use HTML in the tooltips. In order to avoid this, SweenJet provides a improved ToolTipUI that wraps the text into several lines.

```
// Set the tooltip UI
UIManager.put("ToolTipUI", "org.municware.sweenjet.swing.AdvancedToolTipUI");
// Optional setting
UIManager.put("AdvancedToolTipUI.preferredRowCharCount", 25);
```

Menus

Menus are nice too, and there is almost no GUI going without menus. Unfortunately, if you have several menu items from which only some have an icon, the labels are irritatingly no longer aligned. There are two classes that align the menu labels, one for JMenu and one for JPopupMenu.

```
JMenu menu = new AdvancedMenu();
JPopupMenu popupMenu = new AdvancedPopupMenu();
```

Other features

This section covers, for example, how to display splash screens.

Splash screens

Splash screens are very important. They must appear as soon as possible in order to give the user some feedback (let him know that there is something proceeding). And that is how splash screens are created using SweenJet (the source code is taken from Municware GraphTK, but has been modified):

```
/*
 * GraphTkLauncher.java
 */

package org.municware.graphtk.gui;

import java.net.URL;
```

```
import javax.swing.ImageIcon;
import org.municware.sweenjet.swing.GUILauncher;

/**
 * @author Julius Adorf
 */
public final class GraphTkLauncher {

    /** Creates a new instance of GraphTkLauncher */
    private GraphTkLauncher() {
    }

    public static void main(String... args) throws Exception {
        URL iconURL = GraphTkLauncher.class.getResource("/graphtk/banner_short.png");
        GUILauncher launcher = new GUILauncher("org.municware.graphtk.gui.GraphTk",
                "Loading GraphTK", new ImageIcon(iconURL));
        launcher.launch(args);
    }

}
```

The launcher uses reflection, thus mininmizing the number of classes loaded and initialized. So the splash screen appears quite soon after starting the JVM.

Known Limitations

There are no known bugs yet. But there is no software without any bug, so if you find a bug, please report it.

Support

If you are running into problems, please have a look at the FAQ list on our web-site. If this does not solve your problem, please post a support request at our SF website, or send an e-mail to sweenjet(AT)municware.de.

If you find a bug or like to have a specific feature implemented, feel encouraged to contact us and help us to improve this library.

Links

Project home page: http://sweenjet.sourceforge.net/

Project SF home page: http://www.sourceforge.net/projects/sweenjet

Bug reporting: http://sourceforge.net/tracker/?group_id=164063&tid=830218

Support requests: http://sourceforge.net/tracker/?group_id=164063&tid=830219

Feature requests: http://sourceforge.net/tracker/?group_id=164063&tid=830221