Guide to DockingFrames - Common

Benjamin Sigg

January 11, 2008

Abstract

The Common-project is a layer above DockingFrames. It allows to write applications using DF without the need to understand DF. Common does not add new features to DF, Common only combines existing code and reorganizes DF in a way that is easy to use.

1 Introduction

1.1 DockingFrames

What is DF? DF is an open source framework written in Java. It extends Java/Swing with the abbility of "dockable frames". Each frame contains some content, a `JTree` showing a directory or a `Table` showing the results of some game. Each frame has a title and some buttons (like a close-button). The user can move around these frames, the frames will align themself such that there is no unused space between them. There are many ways how frames can be combined to create a new layout.

2 Common

2.1 Basic elements

In the understanding of Common, an ordinary application has one `JFrame`, which is called the "mainframe", showing the content of the application. The content is painted through some `JComponent`s called "panels". Each panel represents some view of the content, for example a texteditor might have one panel for each open document.

Common stands between mainframe and the panels, separating them, and allowing the user to move then panels around.

To do so, each panel gets wrapped into a `FDockable`, such a combination is simply called "dockable". Dockables are then put onto a `FContentArea` which is just a child of the mainframe.

Since there is a complex mechanism between the dockables, a control unit is needed. The control unit is provided by the `FControl`, or just the "controller".

2.2 Creating the controller

The first thing that needs to be done before using Common is to set up the controller. This is done by creating a new object of type `FControl`:

```
1 JFrame frame = new JFrame( "Main–Frame" );
2 boolean secure = false;
3 FControl control = new FControl( frame, secure );
```

Let's have a brief look at the code. Clearly in line `1` the mainframe of the application is created. The controller needs to know the mainframe, it is used as parent of `Windows` which are opened for example when dragging a dockable.

In line `2` it is specified that the application does run in an unsecure environment. An unsecure environment is a normal application. An applet or a webstart-application runs in a secure environment. The controller needs to know that either he can use non-secure optimations (like globally observing all `AWT-Events`) or has to use inefficient workarounds.

Finally in line `3` the controller is created.

2.3 Between mainframe and dockable

After creating mainframe and controller, the layer between mainframe and the dockables has to be set up. A `FContentArea` does the job. The controller grants access to a default-content-area through `FControl.getContentArea()`. If more than one content-area is needed use `FControl.createContentArea(String)` to create additional areas.

```
1 JFrame frame = ...
2 FControl control = ...
3 frame.add( control.getContentArea() );
```

Note line `3`, a content-area is just a `JComponent` and can be added anywhere.

2.4 Wrapping a panel

There are some thoughts needed to create a dockable (to wrap a panel into a `FDockable`). Each kind of panel can fall in one of two categories: the number of instances during the lifetime of an application remains the same, or the number changes.

Those kind of panels whose number does not change, should be wrapped into `FSingleDockable`s, the others in `FMultipleDockable`s. `FSingleDockable` and `FMultipleDockable` are just interfaces, most developers can just use a `DefaultFSingleDockable` or a `DefaultFMultipleDockable`. Only developers interested in writing elements which are parents of dockables need to write new classes and implement the interfaces.

Once a dockable is created, it has to be registered at the controller through the method `FControl.add`. Afterwards it can be made visible `FDockable.setVisible`. Unless otherwise instructed, the controller will then open the dockable at a default location.

2.4.1 Single dockables

A single dockable is an object of type `FSingleDockable`. These dockables are created once by the application, added to the controller and made visible. Then they remain in the memory until they are explicitly removed from the controller, or the application terminates.

Every single dockable needs a unique identifier. This identifier allows the controller to persistant store information about a dockable and later to find the information again.

Example: the list of documents that are currently open in a text editor.

```
1 FControl control = ...
2 FSingleDockable documents = new DocumentList( "myapp-
    document-id" );
3 control.add( documents );
4 documents.setVisible( true );
```

2.4.2 Multiple dockables

A multiple dockable is an object of type `FMultipleDockable`. These dockables are created by the application or the controller, shown for some time and then removed from memory.

Every multiple dockable needs a `FMultipleDockableFactory`, which must have been registered at the controller.

Every time the user loads a previously stored layout, the multiple dockables will be deleted and new instances created. That can be very annoying and disturbing for user and developer. `FWorkingArea`s are designed to prevent such a behavior. They are single dockables, put can also be parent of other dockables.

Example: the documents of a text editor

```
1 FControl control = ...
2 FMultipleFactory factory = new DocumentDockableFactory();
3 control.add( "myapp-document-factory-id", factory );
4
5 FMultipleDockable document = new DocumentDockable(
    factory, "/home/beni/Desktop/file.txt" );
6 control.add( document );
7 document.setVisible( true );
```