1 Docking Frames

2 Overview

2.1 Controller

The class named `bibliothek.gui.DockController` is the most important class of all. The Controller has many purposes: he ensures that one can drag a `Dockable` from one `DockStation` to another. He is also responsible for the title, and the actions. You will see more of this class in the other sections of this document.

What you must remember: whenever you use the docking frames, create a `DockController`, and call the `add`-method with the root-stations. If you don't do that, no titles will be visible, and no darg'n'drop will be available.

```
1  // a root-station
2  DockStation root = new SplitDockStation();
3
4  // a controller
5  DockController controller = new DockController();
6
7  // register the station.
8  controller.add( root );
9
10 // Another station
11 DockStation second = new SplitDockStation();
12
13 // register the second station
14 controller.add( second );
15
16 // now Dockables can be dragged from root
17 // to second, and vice versa.
```

Note Until you want to customize some behaviours, you don't have to care any more about the controller after you registered the roots.

Note Its unimportant if you first register a station and then add some `Dockables` to it, or the other way around. You can even remove a station from a controller and add it to another.

2.2 Dockable

`Dockables` are the `Components` which can be dragged. They have a name and an icon, they also have some controll how their titles are displayed, and the can offer some actions. More about titles and actions will be shown in the "Customizing"-section.

There is an easy to use defaultimplementation: `DefaultDockable`

2.3 DockStation

`DockStation`s are areas where `Dockables` can be dropped. The `Dockables` will have some special behaviour recording to the station on which they are. Many stations are `Dockables` themselfe, so they can be combined in a tree.

Don't forget to register the root-station in a `DockController`. If stations are combinded in a tree, the controller will automatically register the child-stations.

2.3.1 SplitDockStation

This stations behaves like some `JSplitPane`s set into each other. The user can drag the `Dockables` around, and he can also maximze one of the `Dockables`.

2.3.2 StackDockStation

This station is nothing more than an `JTabbedPane`.

2.3.3 ScreenDockStation

This station allows to drag `Dockables` outside a `Window`. They will be displayed on an `JDialog` which has no border.

2.3.4 FlapDockStation

Its children are displayed as buttons. When the user presses one of the buttons, a `Window` will open, and one of the children will be shown.

3 Customizing the docking frames

The docking-frames are designed to be easely modified.

3.1 Titles

Titles are small `Components` used to display the name, the icon and some actions of a `Dockable`.

You can implement an own title by implementing the interface `DockTitle`. Normally a title is his whole lifetime bound to one `Dockable`. However, if you are interested in some cache-mechanisms...
There are two very important methods: `bind` and `undind`. `bind` is invoked befor a title is displayed. In this method you should add your `WhatEverListeners`, and ensure your title displays the current name/icon of the dockable.
You should also have this line of code in the `bind`-method:

```
1 DockActionSource source =
2     dockable.getController().listOffers( dockable );
```

The source is a list of all actions available to the `Dockable`. These actions should be displayed in some way. For example, make a drop-down-menu, or add buttons. You must also invoke the `bind` method of the actions, to ensure they can influence the `Dockable`.

The second method, `unbind` is the opposite of `bind`. Remove all your `WhatEverListeners`, and undind the actions too.

- If you yust want to replace the default-titles, use the `DockController.setTitleFactory`-method. The factory given to this method will be used to create most of the titles.
- How can you ensure that a title is used by a `Dockable`? The story starts in that moment, the parent-`DockStation` of the `Dockable` is registered by the `DockController`. In this moment, the station tries to create a new `DockController.DockTitleVersion` by invoking `DockController.getVersion` (this sets a `DockTitleFactory`). When a version was created, nothing can change its identifier (a string), or replace its `DockTitleFactory`. So the easiest way to use your own titles is to create the version befor a station does. The station will then use your own title-factory.
- If the `DockTitleFactory` is not enough for you, you can override the `getDockTitle`-method of `Dockable`. Doing so, and you have the complet control of every titel for the given `Dockable`. This method is invoked by stations when they have to find a title.

Note It's possible to use `null`-titles. But using no titles will result in a loss of features, there are some stations that may not be able to show a `Dockable` anymore (because one has to click of the title...).

Note Since titles can rapidly be created and destroyed, it's also a good idea to implement a cache-mechanism in the `Dockable`.

3.2 Actions

`DockAction`s are the logic behind buttons and drop-down-menus of the titles. Every action has its own icon, text and its tooltip. Whenever a title for a dockable is binded, the `DockController` will be asked (or at least, should be asked), which actions are available for a the titles's `Dockable`.
You can add a `DockActionListener` to an action, and if the action is invoked, the `actionPerformed`-method is invoked.

There are many sources for actions, and you can change all of them.

- The `Dockable` itself, the method `getActionOffers` returns a list of the own actions.
- The station on which the `Dockable` lies, the method `getDirectActionOffers`

- all stations which are parents of the `Dockable` (this can be a lot, if the `Dockables` themself are stations). The method `getIndirectActionOffers` is important for this.
- By `ActionGuard`s. The guards are registered by an `DockController`. Whenever their `react`-method returns `true`, they will be asked to deliver some actions.
  `ActionGuard`s are the preferred way to add new actions, since you don't have to change the source of `Dockables` or Stations.
- If you want to rearange, or to remove actions, you need more controll. All actions are collected by `ActionOffer`s. You can register your own `ActionOffer` by a `DockController`, and whenever the actions for a `Dockable` are searched, your `ActionOffer` will be asked if he can create the list (the `interested`-method is invoked). If he answers positivly, the `getSource`-method is invoked, and the `ActionOffer` is complitely free how to combine the actions.

3.3 DockableDisplayer

4 Hints, tipps and tricks

There are some tipps that may help customizing and understand the system:

4.1 GlassController

Instances of this class changes the visible-flag of the GlassPane's. The use of this class results in less created objects, and smaller stacks. But the `mouseClicked` event will sometimes be forgotten.

4.2 StationPaint

The StationPaints are used by all default-stations to paint the areas where a `Dockable` will be added. Instances of this interface can be set by the method `DockController.setPaint`, or directly to the stations with a method called `setPaint`.

4.3 onMove()

If you have added some listeners to the controller, or another object used in this system, you may get strange events while a `Dockable` is moved. If you want to know, it the source of the event is a move-operation, call the `onMove()`-method of the `DockController`. It returns `true` only if currently a move-operation is in progress.

4.4 Order matters

The order, when the stations are added to the controller, may become important when a `Dockable` can be dropped on more than one station. Normally, the station added last, has the greatest priority. However, stations can implement the method `compare` and `canCompare` to ensure that they are more/less important than another station.

5 X and Un-X

There are some methods that build pairs which have to be called befor, and after an object is used. These methods are often responsible to register some listener, to make some connections. This list shows some of these pairs, and gives a short explonation:

- `Dockable.bind` This tells a `Dockable`, that a title is shown for it. The title was created by the `Dockable` itself, over the `getDockTitle`-method, which is not allowed to return this instance of a title again until the `unbind` method is called. The `bind`-method has to invoke the `DockableListener.titleBinded`-method. The method is invoked by a `DockStation` which will show the title. The method may be called at any time.
- `Dockable.unbind` The exact opposite of `bind`. Called by a `DockStation`. Can be called at any time.
- `DockTitle.bind` Tells the `DockTitle` that it will be shown as the title of the `Dockable` that has created this instance of the title. The method should setup the title, for the default-implementations this means: using the `DockController` to get the `DockActions`, create and add some `TitleMiniButtons`. The method is called directly from the `DockController`. The method is called if a `Dockable` is registered, or if the `DockableListener.titleBinded`-method of the listener that is added to all `Dockables` is invoked.
- `DockTitle.unbind` The direct opposite of `bind`. Called when a `Dockable` is deregistered of the `DockController` or if the `DockableListener` was notified.
- `DockAction.bind` Tells a `DockAction` that it is shown on a title that refers to a particular `Dockable`. Normally, the method will be called from inside `DockTitle.bind`-method. However, there is no rule that does forbid to call `bind` at any other time.
- `DockAction.unbind` That the direct opposite of `DockAction.bind`.

6 Examples

6.1 No titles for Stations

Normally, every station who is a `Dockable` and sits on another station gets a title. For some applications, the stations don't have to be draggable, and so no titles are needed. This behaviour is simple to implement, just use this code:

```
1  DockController controller = ...
2
3  // The stations have to be removed anyway, but
4  // without title, the user is not able to
5  // remove them
6  controller.setSingleParentRemove( true );
7
8  // Set a new title-factory, that just returns
9  // null for titles of stations
10 controller.setTitleFactory(
11         new DefaultDockTitleFactory(){
12                 public <D extends Dockable & DockStation>
13                         DockTitle createStationTitle(
14                         D dockable, DockTitleVersion version ) {
15
16                         return null;
17                 }
18 });
```