DockingFrames 1.0.2 - FAQ

Benjamin Sigg

February 14, 2008

Contents

Abstract

Some questions that are frequently asked.

1 Writing applications

This section deals with questions that are related to writing code.

1.1 How to write a close-button?

A "close-button" is some button, most times a cross in the upper right edge, that closes a `Dockable` when pressed.

1.1.1 Solution 1: FDockable

When using the common-project, then you can use `DefaultSingleCDockable` or `DefaultMultipleCDockable`. Both `CDockable`s contain a method `setCloseable`. Call that method with `true`.

```
1  DefaultSingleCDockable dockable = ...
2  dockable.setCloseable( true );
```

If you implement the interface `SingleCDockable` or `MultipleCDockable` directly, then ensure that `isCloseable` returns `true`.

1.1.2 Solution 2: DockFrontend

When using a `DockFrontend`, register the `Dockable` at the `DockFrontend` and call `setHideable`.

```
1  DockFrontend frontend = ...
2  Dockable dockable = ...
3
4  frontend.add( dockable, "a_unique_id" );
5  frontend.setHideable( dockable, true );
```

1.1.3 Solution 3: CloseAction

If you use the common-project, then a `DockAction` called `CloseAction` is available. Add the action to the `Dockable`s which should be closeable, for example you could use an `ActionGuard`:

```
1  public class CloseGuard implements ActionGuard{
2      private DockActionSource source;
3
4      public CloseGuard( DockController controller ){
5          DockAction close = new CloseAction( controller );
6          source = new DefaultDockActionSource(
7              new LocationHint(
8                  LocationHint.ACTION_GUARD,
9                  LocationHint.RIGHT_OF_ALL ),
10             close );
11     }
12
13     public boolean react( Dockable dockable ) {
14         return true;
15     }
16     public DockActionSource getSource( Dockable dockable ) {
17         return source;
18     }
19 }
```

And later:

```
1  DockController controller = ...
2  controller.addActionGuard( new CloseGuard( controller ));
```

1.1.4 Solution 4: New DockAction

If `CloseAction` of the common-project can't be used, then a new `DockAction` must be written.

```
1  public class CloseAction extends SimpleButtonAction{
2      public CloseAction( DockController controller ){
3          setText( "Close" );
4          setIcon( new ImageIcon( "close.png" ));
5      }
6
7      @Override
8      public void action( Dockable dockable ) {
9          DockStation parent = dockable.getDockParent();
10         if( parent != null )
11             parent.drag( dockable );
12     }
13 }
```

This action is then used as described in the third solution.

1.2 How do I layout the contents of a SplitDockStation?

The `SplitDockStation` has a complex layout. How can new `Dockable`s be added to `SplitDockStation` such that they have a certain location and size?

1.2.1 Solution 1: SplitDockProperty

A `SplitDockProperty` describes the size and location of a `Dockable` on a `SplitDockStation` by storing the `x`, `y` coordinates, and the `width` and `height`. All the properties are normalized such that they are between 0 and 1.

A client can create new `SplitDockProperty`s and call `drop`:

```
1  SplitDockStation station = ...
2  Dockable alpha = ...
3  Dockable beta = ...
4
5  if( !station.drop( alpha, SplitDockProperty.NORTH ) )
6      station.drop( alpha );
7
8  if( !station.drop( beta, new SplitDockProperty( 0, 0, 1, 1 )))
9      station.drop( beta );
```

A few words to this code: in lines `5` and `8`, the result of `drop` needs to be checked. It is possible, that `SplitDockStation` refuses to add a `Dockable`.

`SplitDockStation` internally has a binary tree in whose leafs the `Dockable`s are stored, and the nodes determine the proportions between the `Dockable`s. Each `drop` adds a new branch into that tree. The `SplitDockProperty` is only a hint where to insert the branch, and will not be stored for later use. So this station does **not** work like a `LayoutManager`, the order in which the `Dockable`s are dropped is important. The first `Dockable` will always get boundaries of `0,0,1,1`.

1.2.2 Solution 2: SplitDockPathProperty

As set earlier, `SplitDockStation` is internally organized as a binary tree. A `SplitDockPathProperty` is the description of the exact location of a branch. It is used like `SplitDockProperty`:

```
1  SplitDockStation station = ...
2  Dockable dockable = ...
3
4  SplitDockPathProperty path = new SplitDockPathProperty();
5  path.add( SplitDockPathProperty.Location.BOTTOM, 0.4 );
6  path.add( SplitDockPathProperty.Location.LEFT, 0.25 );
7  if( !station.drop( dockable, path ))
8      station.drop( dockable );
```

In lines `5,6` a branch to the bottom left edge is created.

1.2.3 Solution 3: SplitDockTree

A `SplitDockTree` is an exact model of the internal binary tree that every `SplitDockStation` has. When calling `dropTree`, all `Dockable`s will be removed from the station, and the new tree will replace the old one.

```
1  SplitDockStation station = ...
2  Dockable[] dockables = ...
3
4  SplitDockTree tree = new SplitDockTree();
5  tree.root(
6      tree.horizontal(
7          tree.vertical(
8              dockables[0],
9              dockables[1],
10             0.4 ),
11         tree.vertical(
12             tree.put(
13                 dockables[2] ),
14             tree.put(
15                 dockables[3],
16                 dockables[4] ) )
17     )
18 );
19 station.dropTree( tree );
```

Note that several `Dockable`s can be put at the same location as shown in lines `14-16`.

1.2.4 Solution 4: SplitDockGrid

A `SplitDockGrid` is an algorithm that takes several `Dockable`s and their desired location, and creates a `SplitDockTree` that matches these locations as good as possible.

The use is straight forward:

```
1  SplitDockStation station = ...
2  Dockable[] dockables = ...
3
4  SplitDockGrid grid = new SplitDockGrid();
5  grid.addDockable( 0.0, 0.0, 0.5, 0.5, dockables[0] );
6  grid.addDockable( 0.0, 0.5, 0.5, 0.5, dockables[1] );
7  grid.addDockable( 0.5, 0.0, 0.5, 0.5, dockables[2] );
8  grid.addDockable( 0.5, 0.5, 0.5, 0.5, dockables[3] );
9  station.dropTree( grid.toTree() );
```

There is also the possibility to model the layout with a `String`:

```
1  SplitDockStation station = ...
2  Dockable[] dockables = ...
3
4  Map<String, Dockable[]> map = new HashMap<String, Dockable[]>();
5  for( int i = 0; i < dockables.length; i++ ){
6      map.put(
7          String.valueOf( i ),
8          new Dockable[]{ dockables[i] });
9  }
10
11 String layout =
12     "0022\n"+
13     "0022\n"+
14     "1133\n"+
15     "1133";
16
17 SplitDockGrid grid = new SplitDockGrid( layout, map );
18 station.dropTree( grid.toTree() );
```

The layout is defined in lines `12-15`, just image a raster where the characters tell which `Dockable` should overlap a given cell.

1.2.5 Solution 5: Start and store

If your application is able to store the layout, then just start the application, make the layout by hand, and store the layout.

The common-project or `DockFrontend` can help you storing one or several layouts.