



# Java Desktop Integration Components (JDIC)

## J.1 Introduction

The **Java Desktop Integration Components (JDIC)** are part of an open-source project aimed at allowing better integration between Java applications and the platforms on which they execute. Some JDIC features include:

- interacting with the underlying platform to launch native applications (such as web browsers and e-mail clients)
- displaying a splash screen when an application begins execution to indicate to the user that the application is loading
- creating icons in the system tray (also called the taskbar status area or notification area) to provide access to Java applications running in the background
- registering file-type associations, so that files of specified types will automatically open in corresponding Java applications
- creating installer packages, and more.

The JDIC homepage ([jdic.dev.java.net/](http://jdic.dev.java.net/)) includes an introduction to JDIC, downloads, documentation, FAQs, demos, articles, blogs, announcements, incubator projects, a developer's page, forums, mailing lists, and more. Java SE 6 includes some of the features mentioned above. We discuss several of these features here.

## J.2 Splash Screens

Java application users often perceive a performance problem, because nothing appears on the screen when you first launch an application. One way to show a user that your program is loading is to display a **splash screen**—a borderless window that appears temporarily

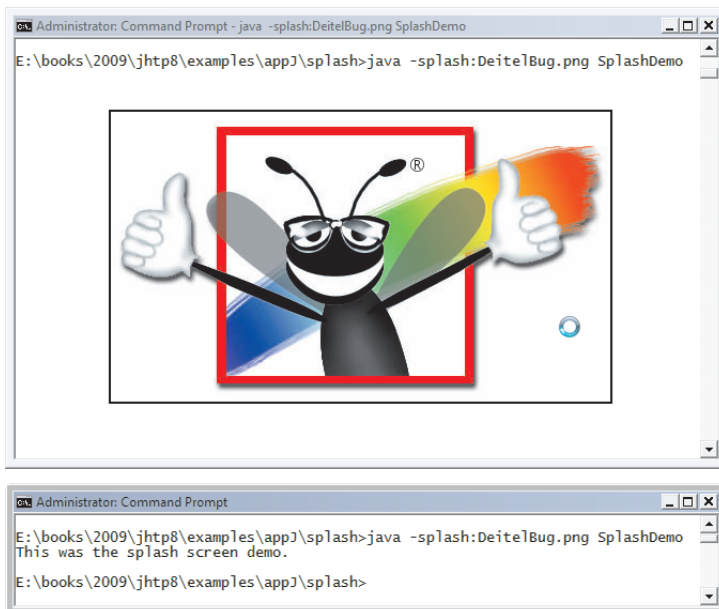
while an application loads. Java SE 6 provides the new command-line option **-splash** for the java command to accomplish this task. This option enables you to specify a PNG, GIF or JPG image that should display when your application begins loading. To demonstrate this new option, we created a program (Fig. J.1) that sleeps for 5 seconds (so you can view the splash screen) then displays a message at the command line. The directory for this example includes a PNG format image to use as the splash screen. To display the splash screen when this application loads, use the command

```
java -splash:DeitelBug.png SplashDemo
```

```

1  // Fig. J.1: SplashDemo.java
2  // Splash screen demonstration.
3  public class SplashDemo
4  {
5      public static void main( String[] args )
6      {
7          try
8          {
9              Thread.sleep( 5000 );
10             } // end try
11             catch ( InterruptedException e )
12             {
13                 e.printStackTrace();
14             } // end catch
15
16             System.out.println(
17                 "This was the splash screen demo." );
18         } // end method main
19     } // end class SplashDemo

```



**Fig. J.1** | Splash screen displayed with the `-splash` option to the java command.

Once you've initiated the splash screen display, you can interact with it programmatically via the **SplashScreen** class of the `java.awt` package. You might do this to add some dynamic content to the splash screen. For more information on working with splash screens, see the following sites:

```
java.sun.com/developer/technicalArticles/J2SE/Desktop/javase6/
    splashscreen/
java.sun.com/javase/6/docs/api/java/awt/SplashScreen.html
```

### J.3 Desktop Class

Java SE 6's new **Desktop** class enables you to specify a file or URI that you'd like to open using the underlying platform's appropriate application. For example, if Firefox is your computer's default browser, you can use the Desktop class's `browse` method to open a website in Firefox. In addition, you can open an e-mail composition window in your system's default e-mail client, open a file in its associated application and print a file using the associated application's print command. Figure J.2 demonstrates the first three of these capabilities.

The event handler at lines 86–116 obtains the index number of the task the user selects in the `tasksJComboBox` (line 89) and the `String` that represents the file or URI to process (line 90). Line 92 uses Desktop static method **isDesktopSupported** to determine whether class Desktop's features are supported on the platform on which this application runs. If they are, line 96 uses Desktop static method **getDesktop**, to obtain a Desktop object. If the user selected the option to open the default browser, line 101 creates a new URI object using the `String` input as the site to display in the browser, then passes the URI object to Desktop method **browse** which invokes the system's default browser and passes the URI to the browser for display. If the user selects the option to open a file in its associated program, line 104 creates a new `File` object using the `String` input as the file to open, then passes the `File` object to Desktop method **open** which passes the file to the appropriate application to open the file. Finally, if the user selects the option to compose an e-mail, line 107 creates a new URI object using the `String` input as the e-mail address to which the e-mail will be sent, then passes the URI object to Desktop method **mail** which invokes the system's default e-mail client and passes the URI to the e-mail client as the e-mail recipient. You can learn more about class Desktop at

```
java.sun.com/javase/6/docs/api/java/awt/Desktop.html
```

---

```
1  // Fig. J.2: DesktopDemo.java
2  // Use Desktop to launch default browser, open a file in its associated
3  // application and an email in the default email client.
4  import java.awt.Desktop;
5  import java.io.File;
6  import java.io.IOException;
7  import java.net.URI;
8
9  public class DesktopDemo extends javax.swing.JFrame
10 {
```

---

**Fig. J.2** | Use Desktop to launch the default browser, open a file in its associated application and compose an e-mail in the default e-mail client. (Part I of 3.)

---

```

11 // constructor
12 public DesktopDemo()
13 {
14     initComponents();
15 } // end DesktopDemo constructor
16
17 // To save space, lines 20-84 of the NetBeans autogenerated GUI code
18 // are not shown here. The complete code for this example is located in
19 // the file DesktopDemo.java in this example's directory.
20
85 // determine selected task and perform the task
86 private void doTaskJButtonActionPerformed(
87     java.awt.event.ActionEvent evt)
88 {
89     int index = tasksJComboBox.getSelectedIndex();
90     String input = inputJTextField.getText();
91
92     if ( Desktop.isDesktopSupported() )
93     {
94         try
95         {
96             Desktop desktop = Desktop.getDesktop();
97
98             switch ( index )
99             {
100                 case 0: // open browser
101                     desktop.browse( new URI( input ) );
102                     break;
103                 case 1: // open file
104                     desktop.open( new File( input ) );
105                     break;
106                 case 2: // open email composition window
107                     desktop.mail( new URI( input ) );
108                     break;
109             } // end switch
110         } // end try
111         catch ( Exception e )
112         {
113             e.printStackTrace();
114         } // end catch
115     } // end if
116 } // end method doTaskJButtonActionPerformed
117
118 public static void main(String[] args)
119 {
120     java.awt.EventQueue.invokeLater(
121         new Runnable()
122         {
123             public void run()
124             {
125                 new DesktopDemo().setVisible(true);
126             }
127         }
128     );
129 }

```

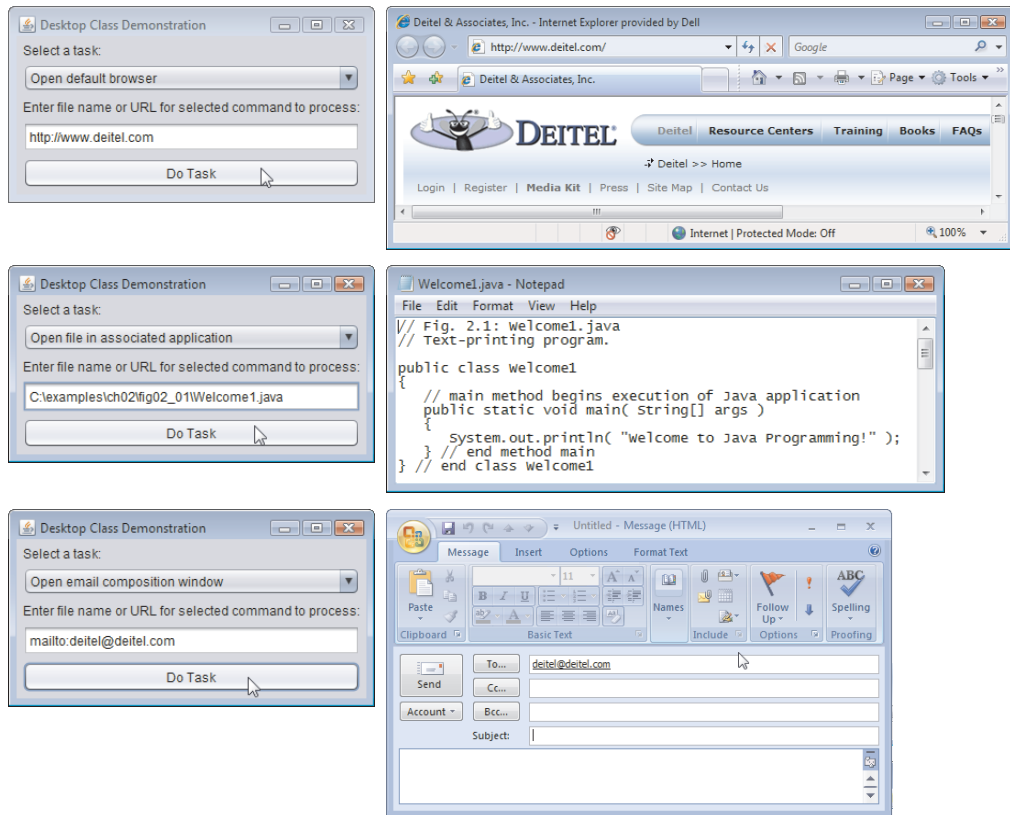
---

**Fig. J.2** | Use Desktop to launch the default browser, open a file in its associated application and compose an e-mail in the default e-mail client. (Part 2 of 3.)

```

127     }
128     );
129 } // end method main
130
131 // Variables declaration - do not modify
132 private javax.swing.JButton doTaskJButton;
133 private javax.swing.JLabel inputJLabel;
134 private javax.swing.JTextField inputJTextField;
135 private javax.swing.JLabel instructionLabel;
136 private javax.swing.JComboBox tasksJComboBox;
137 // End of variables declaration
138 }

```



**Fig. J.2** | Use Desktop to launch the default browser, open a file in its associated application and compose an e-mail in the default e-mail client. (Part 3 of 3.)

## J.4 Tray Icons

**Tray icons** generally appear in your system's system tray, taskbar status area or notification area. They typically provide quick access to applications that are executing in the background on your system. When you position the mouse over one of these icons, a tooltip appears indicating what application the icon represents. If you click the icon, a popup menu appears with options for that application.

Classes `SystemTray` and `TrayIcon` (both from package `java.awt`) enable you to create and manage your own tray icons in a platform independent manner. Class **`SystemTray`** provides access to the underlying platform's system tray—the class consists of three methods:

- static method **`getDefaultSystemTray`** returns the system tray
- method **`addTrayIcon`** adds a new `TrayIcon` to the system tray
- method **`removeTrayIcon`** removes an icon from the system tray

Class **`TrayIcon`** consists of several methods allowing users to specify an icon, a tooltip and a pop-up menu for the icon. In addition, tray icons support `ActionListeners`, `MouseListeners` and `MouseMotionListeners`. You can learn more about classes `SystemTray` and `TrayIcon` at

```
java.sun.com/javase/6/docs/api/java/awt/SystemTray.html
java.sun.com/javase/6/docs/api/java/awt/TrayIcon.html
```

## J.5 JDIC Incubator Projects

The JDIC Incubator Projects are developed, maintained and owned by members of the Java community. These projects are associated with, but not distributed with, JDIC. The Incubator Projects may eventually become part of the JDIC project once they have been fully developed and meet certain criteria. For more information about the Incubator Projects and to learn how you can setup an Incubator Project, visit

```
jdic.dev.java.net/#incubator
```

## J.6 JDIC Demos

The JDIC site includes demos for `FileExplorer`, the browser package, the `TrayIcon` package, the `Floating Dock` class and the `Wallpaper API` (`jdic.dev.java.net/#demos`). The source code for these demos is included in the JDIC download (`jdic.dev.java.net/servlets/ProjectDocumentList`). For more demos, check out some of the incubator projects.